# Stream Processing with Adaptive Edge-Enhanced Confidential Computing

Yuqin Yan
yuqin.yan@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Pritish Mishra
pritish@cs.toronto.edu
University of Toronto
Toronto, Ontario, Canada

Wei Huang
wh.huang@mail.utoronto.ca
University of Toronto
Toronto, Ontario, Canada

Aastha Mehta
aasthakm@cs.ubc.ca
University of British Columbia
Vancouver, British Columbia, Canada

Oana Balmau
oana.balmau@cs.mcgill.ca
McGill University
Montreal, Quebec, Canada

David Lie
david.lie@utoronto.ca
University of Toronto
Toronto, Ontario, Canada

## ABSTRACT

Stream processing is becoming increasingly significant in various scenarios, including security-sensitive sectors. It benefits from keeping data in memory, which exposes large volumes of data in use, thereby emphasising the need for protection. The recent development of confidential computing makes such protection technologically feasible. However, these new hardware-based protection methods incur performance overhead. Our evaluation shows that replacing legacy VMs with confidential VMs to run streaming applications incurs up to 8.5% overhead on the throughput of the queries we tested in the NEXMark benchmark suite. Pursuing specialised protection for broader attacks, such as attacks at the edge with more physical exposure, can push this overhead further. In this paper, we propose a resource scheduling strategy for stream processing applications tailored to the privacy needs of specific application functions. We implement this system model using Apache Flink, a widely-used stream processing framework, making it aware of the underlying cluster's protection capability and scheduling the application functions across resources with different protections tailored to the privacy requirements of an application and the available deployment environment.

## CCS CONCEPTS

• **Security and privacy → Distributed systems security**; • **Information systems → Stream management**.

## KEYWORDS

Security, Data Streaming, Trusted Execution Environment, Confidential Computing, Stream Processing Framework

**ACM Reference Format:**
Yuqin Yan, Pritish Mishra, Wei Huang, Aastha Mehta, Oana Balmau, and David Lie. 2024. Stream Processing with Adaptive Edge-Enhanced Confidential Computing. In *7th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '24), April 22, 2024, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3642968.3654819


*EdgeSys '24, April 22, 2024, Athens, Greece*

ACM ISBN 979-8-4007-0539-7/24/04…$15.00
https://doi.org/10.1145/3642968.3654819

## 1 INTRODUCTION

Stream processing is an increasingly important computing paradigm that facilitates on-the-fly data analysis. It plays a crucial role in enabling businesses to adopt an agile and responsive operational approach [17] and can be used in several critical edge applications, such as ML inference, real-time detection of fraudulent credit transactions [14], and monitoring patient health data. A key feature of stream processing is the ability to place computation close to data, with the objective of optimising for bandwidth, latency, and computational costs. However, many stream processing applications also handle security-sensitive data and code; thus, ensuring the confidentiality of data and the integrity of the computation can also be an important concern.

To alleviate the security concerns of applications hosted in untrusted and third-party platforms (e.g., cloud platforms), CPU vendors have proposed hardware primitives for *confidential computing*. For instance, AMD's SEV [9], SEV-SNP [4] and SEV-TIO, Intel's TDX [6], and ARMv9's CCA [13] extensions all provide some form of a *trusted execution environment* (TEE), which provides hardware-assisted cryptographic protection for code and data executing on these platforms. Further, many of the latest iterations of these mechanisms enable confidential VMs to run on top of an untrusted hypervisor, enabling the protection of entire legacy system images through a "lift-and-shift" approach.

However, these TEE mechanisms come with a cost. The encryption and decryption of memory, as well as checks against tampering for integrity, impose runtime overheads on workloads executing in protected TEE environments. Further, transitions in and out of TEE execution modes may also incur overheads. Finally, current hardware-based TEEs do not provide complete protection and, in particular, may be vulnerable to side-channel and physical attacks, which would require more sophisticated mitigation mechanisms that impose even more overhead.

This is particularly relevant to edge services, where an *edge datacenter* [15] may be less secure against physical attack than a cloud datacenter. Current server-side TEEs have been designed with cloud datacenter threat models in mind and do not take into account physical attacks. For example, they do not account for attacks where an attacker can snoop the memory bus [10] or replace the DRAM

with an older snapshot. Defending these is possible with additional measures, such as using secured or encrypted memory [3, 7], but with additional performance overhead or hardware cost.

Thus, a key question is, "How should the security and privacy objectives be incorporated with the existing objectives of bandwidth, latency and compute cost when deploying stream processing systems?" Rather than being a universal cost imposed on every operator of a streaming application, a more nuanced approach could examine and assign a security requirement to each operator based on the privacy requirements of the data that the operator operates on and produces. This requirement would then be incorporated into the scheduling decisions for the placement of the operators—operators with a security requirement may then be scheduled on nodes that provide a commensurate level of security protection using one or more of the security mechanisms we have mentioned so far. This would enable operators with a lower or no security requirement to be scheduled on faster or less specialised, commodity hardware, while operators with a higher security requirement can be scheduled on slower or more specialised secure hardware.

In this work, we make the following contributions:

- We propose a security measure that can be used as an additional factor for making scheduling decisions in stream computing systems.
- We prototype a framework that uses this measure by extending task slot scheduling strategies in Flink [5], a representative stream processing framework.

The remainder of this paper is organised as follows. In Section 2, we describe the general design of stream processing and the possible cost of applying confidential computing. Then we present our system design in Section 3 with motivating examples of existing benchmarks. We evaluate the performance in realistic environments with different configurations in Section 5. Finally, we conclude this paper with plans for future work in Section 6.

# 2 BACKGROUND & MOTIVATION

## 2.1 Stream Processing Overview

Stream processing frameworks use a dataflow execution model where an **application** is represented as a directed acyclic graph (DAG) whose vertices represent operators and edges represent the data streams connecting the operators. Each operator encapsulates programming logic, such as data filtering, stream aggregation, or function evaluation. Users use the framework to specify a *logical plan* of the operators and the data flows between the operators.

Resource scheduling in stream processing frameworks allows applications to exploit horizontal/vertical scalability such that data streams are partitioned and processed at different locations simultaneously and independently. Operators in a given topology can be scaled dynamically with a given parallelism degree, generating multiple replicas to execute simultaneously on top of distributed resources. The logical plan submitted by the user is converted into a *physical plan* that defines the placement of these replicas on available computing resources, and a *scheduler* uses this plan to spawn these replicas dynamically.

Flink [5] is a representative scale-out stream processing framework supporting such rich stream processing semantics. Extending the standard resource scheduling abstraction, Flink pre-allocates the computing resources for a given streaming application but offers additional flexibility by allowing multiple operators to share a single computing resource.

## 2.2 Confidential Computing and the Cost

In traditional virtualisation, the hypervisor has full control over VMs, including direct access to and manipulation of a VM's memory. Confidential computing establishes a secure environment for VMs, safeguarding them against potentially untrustworthy hypervisors by preventing unauthorised access and modification. Through hardware assistance, the hypervisor's ability to access and modify a VM's private memory and critical state is curtailed. VMs can selectively share their state with the hypervisor. This architecture safeguards the confidentiality and integrity of data within a confidential VM, ensuring that sensitive information remains protected from unauthorised access or tampering.

Confidential computing introduces various types of overhead during VM execution. For example, confidential computing encrypts and decrypts data for confidentiality, though this cost is mitigated with hardware acceleration. Similarly, integrity protection also incurs overhead during state changes such as memory writes and TLB installations. Finally, protection domain switches in and out of confidential computing mode incur overhead due to an increased number of checks by trusted security modules and the use of bounce buffers for I/O events can impose extra costs on I/O-intensive workloads [11].

In addition, the cost can vary based on the hardware design, the computing environment, and level of desired security. For instance, despite advancements in hardware, SEV-SNP is susceptible to ciphertext side channel attacks [12]. This vulnerability arises from the hypervisor's ability to access encrypted memory combined with the deterministic nature of ciphertext when a VM writes the same plaintext to the same physical address. Such a scenario undermines the security of legacy applications, and software mitigation is required [16]. Conversely, Intel TDX mitigates this issue by allowing the hypervisor only to read a fixed NULL value for memory belonging to a Trusted Domain (TD), effectively isolating the VM's memory from software access. However, it maintains extra metadata information, and TDX-compatible hardware has not yet achieved widespread availability, limiting its adoption.

With its potential for physical exposure, the edge computing scenario opens up additional attack vectors that could compromise confidentiality and integrity guarantees. Fortunately, there have been proposals aimed at enhancing protection at the DRAM side [3, 7], extending the trusted computing boundary at the cost of maintaining and checking extra information during memory events with advanced DRAM technologies. These solutions leverage advanced DRAM technologies to maintain and verify additional information during memory events, enhancing security at the expense of increased overhead. However, it poses a challenge of availability, as the complexity of these technologies makes them more difficult to bring to market.

Side-channel attacks and microarchitecture vulnerabilities remain persistent concerns for shared trusted hardware, a situation exacerbated in the post-Spectre and Meltdown landscape. Researchers

have shown that cross-VM-boundary attacks exploiting microarchitecture vulnerabilities are feasible, and the confidentiality of VMs is at risk due to the non-encryption of data inside the enclaves. A common strategy involves isolating VMs by preventing them from sharing the physical cores. However, implementing this strategy effectively requires disabling hyperthreading at the BIOS level due to the hypervisor's untrusted status in confidential computing's threat model and its privileged ability to manipulate the system's scheduler. While mitigating security risks, it can significantly impact a system's multi-core performance.

## 3 DESIGN

The objective of our design is to extend existing stream processing scheduling mechanisms to take into account - a) the protection requirements of data, which is determined by the sensitivity of the exposed data, and b) the security level of computational nodes, which can be determined by their hardware and software features, as well as their physical security. To achieve this, we propose that this extension take into account three properties for nodes and data types in streaming applications: (1) a set of protection capability descriptions (Section 3.1); (2) a label indicating the set of protection capabilities each node in a stream processing cluster can provide (Section 3.2); (3) a protection-aware way of writing the stream applications (Section 3.3).

### 3.1 Protection Capability Description and Protection Profile

The protection capability is described by **protection levels** and **security concerns**. Table 1 gives an example of a confidential-computing-centric protection capability description. The protection levels in this table represent the support level for confidential computing. For example, for AMD EPYC processors, only SEV with both ES and SNP extensions enabled can be placed in Level 3, as only the SNP extension supports integrity protection from a malicious hypervisor. The protection level can be verified via *remote attestation*, a security process commonly provided as a component of confidential computing infrastructure, used to verify that a remote computer system is in a specific state, including the hardware and firmware information.

The security concerns in the table describe issues that have not been completely or commonly resolved by current confidential computing designs. If available, security can be enhanced with certain practices. The values are a nullable boolean, signifying whether a concern is considered to be mitigated on the resource node. *NULL* value indicates a trustworthy measurement cannot be performed. This value is assigned to the node through remote attestation and other information such as the instance type, its level in the hierarchy (cloud, edge, etc.), or `CPUID` obtained with trusted procedures such as `CPUID` filtering [4].

Mitigation of security concerns comes with performance cost or availability issues. Cipherfix [16] mitigates ciphertext side-channel issues through runtime instrumentation, imposing a significant mean performance overhead ranging from 2.4x to 17.5x across various cryptographic libraries. In edge computing contexts, SecDDR [7] offers a novel approach to countering physical replay attacks by incorporating security logic into the ECC chip, necessitating advancements in processing-in-memory technologies. Meanwhile, InvisiMem [3] mitigates both address side-channel and memory timing vulnerabilities, ensuring data freshness with a reasonable performance overhead of 15.21%. However, it depends on a packetised interface rather than the traditional DDR interface, which might limit its broader applicability.

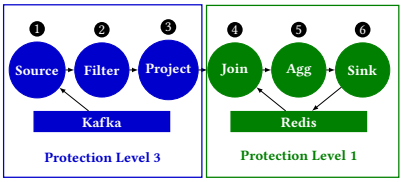


**Figure 1: Yahoo! Streaming Benchmark: The job graph and the expected partitioning of different protection levels**

A protection profile is derived by specifying the protection level and the values of the security concerns.

### 3.2 Protection Provisioning in Cluster

We assume the controller containing a cluster's resource manager is a trusted service in a secure environment. For example, it can be deployed on self-managed machines with trusted hypervisor software and physical protection. The cluster controller registers the computing nodes into the processing cluster with a protection profile describing the **maximum** protection a node can provide. The protection profile specifies the maximum protection level the node can be allocated to and assigns values for each entry in the list of security concerns according to the description in Section 3.1.

### 3.3 Protection Request in Application

To utilise the protection that the worker nodes in a cluster can provide, the application needs to indicate the protection it requires to deal with potential threats. For each operator in the *logical plan*, the application developer can provide each operator with a protection profile, describing the **minimum** protection level and the accepted values for each entry of the security concerns, taking into consideration the sensitivity and privacy needs of the data and the potential threats of the processing environment.

### 3.4 Scheduling

After an application is submitted to the cluster, each operator spawns parallel replicas according to the parallelism value specified for the operator. Each replica inherits the protection profile from the operator. During scheduling, the cluster scheduler aims to satisfy the constraint of the protection request specified in the protection profile: the protection level provided by a node must be larger than or equal to an operator's request, and the security concerns in the node's protection profile must be the values the task can accept. The scheduling cannot succeed if no node can satisfy the protection request.

**Table 1: An example of a confidential-centric protection capability description: Each node in a cluster must be at a protection level and declare its security concerns. The security concerns have three possible values: yes, no, and unspecified. Refer to Section 3.1 for details.**

| Protection Level | Security Description |
|---|---|
| 1 | Legacy VMs without confidential computing support, or no attestation report is provided |
| 2 | The attestation report indicates key security-related confidential computing extensions are not enabled, such as a SEV-enabled VM without SEV-ES or SEV-SNP. A confidential-computing-enabled VM with outdated firmware also falls into this level. |
| 3 | The attestation report indicates the security-related confidential computing extensions are fully enabled (SEV with SEV-ES and SEV-SNP on the AMD platform, or TDX-enabled on the Intel platform). The firmware is up-to-date. |

| Security Concerns | Security Description | Performance/Availability Implication |
|---|---|---|
| Same-core microarchitecture security | The attestation report indicates whether the hyperthreading is turned off. If so, it mitigates generic same-core side-channel and microarchitecture attacks. | Performance degradation on multi-core performance by halving the number of physical threads. |
| Software ciphertext side channel | It indicates whether a node is vulnerable to the ciphertext side channel by software access from the hypervisor. For example, the current SEV has this side channel, but TDX does not. | It requires code auditing or runtime mitigation or prevents deployment onto some architecture if such concern exists, such as SEV. |
| Memory bus snooping (edge) | The attacker can monitor the bus signals. For example, the ciphertext side channel exists even with the read-blocking guarantee provided by TDX, and the data is encrypted. | Advanced DRAM design can mitigate this issue at a relatively low-performance cost but has manufacturing and availability issues. |
| Replay attack with physical access (edge) | The attacker may either replace the DRAM with a stale snapshot or replay a recorded older packets. | |

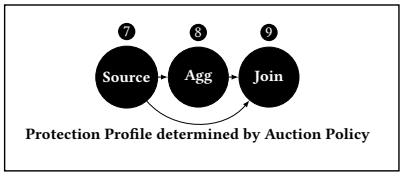


**Figure 2: NEXMark Q7: The protection level required by this query is determined by the auction policy.**

## 3.5 Example: Yahoo! Streaming Benchmark

The Yahoo! streaming benchmark [2] is an advertising application whose logical plan is presented as a DAG in Figure 1. The application first ingests click events from a Kafka input stream. A click event contains a user ID, a page ID, an advertisement ID, the event type, the timestamp of the event, and the IP address of the event source (❶.) The application then filters the events with the type "view" and discards the events with other types (❷) and projects them so that only the advertisement ID and the event timestamps are further processed (❸). The advertisement ID is then transformed into a campaign ID (❹) through a join operation that looks for the campaign ID that the advertisement ID belongs to. The tuple (campaign ID, timestamp) is then fed into a time window aggregation node (❺) to count the number of campaigns viewed in a time window. At last, it updates the campaign counts in Redis periodically until the window is closed (❻).

We assume that the attacker is at a privileged level, is honest-but-curious, and keeps monitoring and recording the data during processing but does not tamper with the integrity of computing. This is a rational scenario, considering an attacker may wish to stealthily extract sensitive information. Although the protection policy could be application-dependent, we chose one that protects the information of users, as opposed to that of the advertisement campaign from potential monitoring. Consequently, the source node, the filter node, and the projection nodes request protection level 3 with confidential computing enabled, and the following processing request protection level 1. The expected splitting is displayed in Figure 1.

## 3.6 Example: NEXMark Benchmarks

The NEXMark benchmark [1] simulates an auction scenario with three types of events: person, auction, and bids. Each event type is described with several attributes. We found that the bid event is the major type being processed in most of the queries in the benchmark suite, which does not exhibit strict confidentiality requirements in real life: there are scenarios where the prices of bids are publicly revealed and others where confidentiality is required. Q7 (Figure 2) computes the highest bid per period. It ingests the bid events as the input stream (❼), fed into a windowed aggregator which outputs the max price during a period (❽), and the output joins with the bid price again to retrieve the other bid information (❾). This query only processes the bid event, and its protection request is determined by auction policy. With the policy information, a flexible protection profile for requesting protection can be generated and constrain the operator scheduling.

# 4 IMPLEMENTATION

We implemented the design based on Apache Flink enabling the Flink cluster to provide protection according to a Flink application's

request. We create a class, `ProtectionProfile`, containing the attributes reflecting the definitions in the protection capability description (Section 3.1). We attach this as a part of the `Resource` class. During the cluster initialisation, when a TaskManager (worker) joins, it reports to the cluster's resource manager with `Resource` information attached with `ProtectionProfiles`. For simplicity (as we focus more on the scheduling part), we did not implement the verification procedure, where the JobManager (controller) triggers the remote attestation and parses the report submitted by the TaskManager. We leave it as a future work.

For Flink application development, we implemented support for the DataStream API, specifying the application logic in the flavour of a Java application instead of a table-based SQL interface. We utilise Flink's *slot-sharing group* to achieve confidential-computing-aware development and scheduling. A slot-sharing group specifies which operators can be collocated in the same execution slot. When constructing a slot-sharing group, the application developer can attach a `ResourceProfile` of the slot-sharing group to indicate the resource requirement for executing the tasks inside the group, containing the requested protection profile as well. We add an interface for specifying the `ProtectionProfile` for each operator. During scheduling, the scheduler compares the protection profile of the slot ($PP_{slot}$) and the protection profile of the slot-sharing group ($PP_{ssg}$) while checking whether a slot can satisfy the protection request of a slot-sharing group, which holds iff the following conditions hold: (1) $PP_{slot}$.`protection_level` $\geq$ $PP_{ssg}$.`protection_level` (2) $PP_{slot}$.`security_concern`$_i$ $\subseteq$ $PP_{ssg}$.`security_concern`$_i$, as Section 3.4 indicates.

## 5 EVALUATION

Our environment of evaluation is an AMD 3rd EPYC 7543 server with 256 GiB memory. We used the `snp-latest` branch of the `AMDESE/AMDSEV` repository on GitHub and compiled kernel images with git commit `6b293770dac2` for both the host and the guests.

Our evaluations are done within the following protection environment: ① at protection level 1, where the SEV is not enabled; ② at protection level 3 with AMD SEV-SNP enabled; and ③ at protection level 3, mitigating the concern of same-core microarchitecture security by disabling SMT when booting up the host and only provisioning half of the physical threads for the guest VM. In all protection environments, the disks are LUKS-encrypted.

It is worth noting that our evaluation was conducted only within a limited combination of protection environments. This limitation stems from the constraint that the DRAM designs detailed in Section 3.1 have yet to enter production and market, preventing us from conducting a concrete evaluation.

***Experiment 1: Latency.*** Yahoo! streaming benchmark has an aggregation window operator. It defines the latency of this query with the attributes of this window (the window size, the last update after the window closes, and the start time of the window) as it is the central operation and produces the results to the sink. For every window opened at time $T_{open}$, records arrive in this window and contribute to the aggregation results until the window is finally closed. When the window is closed, the final aggregation result is written into the Redis sink at time $T_{last\_update}$. Assume the window length is $t_{window}$, the official definition of the latency

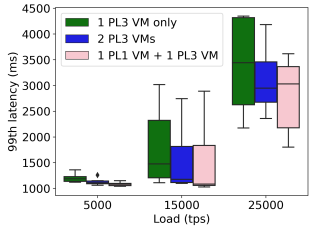


**Figure 3: 99 percentile latency distribution of 8 times of running Yahoo streaming benchmark where TM stands for TaskManager, PL stands for protection level. Load is measured in the number of input events per second.**

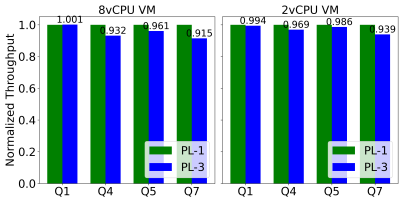


**Figure 4: Comparison of normalised throughput results for running NEXMark benchmark**

$\ell$ is $\ell = (T_{last\_update} - T_{open}) - t_{window}$. The results are displayed in Figure 3, showing the distribution of the $99^{th}$ percentile of the latency under different loads (the number of records generated per second). We evaluate three *configurations*, where each VM has one vCPU: (1) 1 VM at environment ② with all processing components (green), (2) 2 VM at environment ② with component placement shown in Figure 1, except for the fact that they are placed in two VMs with the same protection environment (blue) (3) 1 VM at environment ① and 1 VM in ② with component placement displayed in Figure 1 (pink). The results in Figure 3 show that adding computing resources by splitting operators can have the benefit of reducing latency, especially when the load is high, and the use of tailored protection can not only relax the constraints to make the scheduler have more candidates VMs to schedule but also have the potential of reducing latency by placing partial computing in an environment with less overhead of accessing memory. However, adding resources in the way of the transition from Configuration 1 to 2 (and 3 as well) requires changing local network traffic to cross-VM traffic at the stream represented as the edge between operator ❸ and ❹ in Figure 1, whose cost can be more significant in a real-life setting with VMs located on different physical machines.

***Experiment 2: Throughput.*** We use the NEXMark benchmark suite to evaluate the throughput under different protection capacities. We select Q1 as a representative instance of a simple stateless operator, `MAP`, and Q4, Q5, and Q7 as instances of stateful operators with large operator states. The data source generates the input data in memory and uses the Blackhole connector [8] as the sink to remove the effect of the source and the sink's performance. We remove the checkpointing mechanism by setting the interval to a large value so it is never triggered.

*Experiment 2.1* We have three worker VMs. Each of the three VMs has one TaskManager with one task slot, and the total parallelism